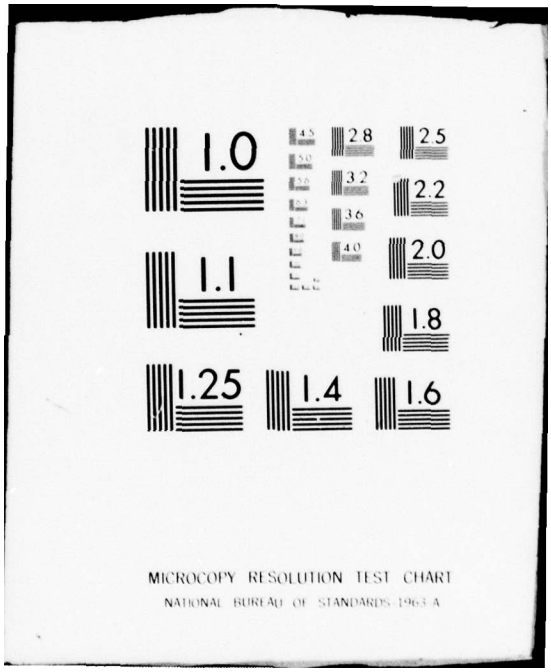
AD-A066 993 WHARTON SCHOOL PHILADELPHIA PA DEPT OF DECISION SCIENCES F/G 9/2
A DATABASE INTERFACE TO WAND FOR THE NETWORK ALERTER SERVICE.(U)
DEC 78 J S RIBEIRO N00014-75-C-0462
UNCLASSIFIED 78-11-02 NL

| OF |
AD A066993

END
DATE FILMED
6-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

Wharton

Department of Decision Sciences

LEVEL

12

AD A066993

DDC FILE COPY



University of
Pennsylvania
Philadelphia PA 19104



79 04 04 [illegible]

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

| | |
|---|---|
| 1. REPORT NUMBER: 78-11-02 | 2. GOVT ACCESSION NO. |
| 3. RECIPIENT'S CATALOG NUMBER | |
| 4. TITLE (and Subtitle): A Database Interface to WAND for the Network Alerter Service. | 5. TYPE OF REPORT & PERIOD COVERED: technical report. Apr 78 — Mar 79 |
| | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s): James Steven ~~Ribiero~~ Ribeiro | 8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0462 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Decision Sciences, The University of Pennsylvania, Philadelphia, PA 19104 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Task NR049-272 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research | 12. REPORT DATE: November 78 |
| | 13. NUMBER OF PAGES: 48 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) | 15. SECURITY CLASS. (of this report): unclassified |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

11 Dec 78

12 53 p.

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Distribution unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Alerting, monitoring, database management, computer networks, dynamic database, network alerter service.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Database Interface (DBI) is responsible for monitoring conditions on a data base managed by a particular type of Database Management System (DBMS). At the Wharton School there are several large databases which are managed by WAND, a CODASYL-like DBMS. This thesis outlines the design and implementation of a DBI for WAND. The DBI will be integrated into the Network Alerter Service (NAS) which is being designed and implemented as a general user service for ARPANET users. The NAS will allow the monitoring of databases at various sites on the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601 |

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408 757

(cont)



ARPANET, for conditions of interest to the user. The WAND DBI is capable of efficiently monitoring these conditions or alerters, and responding appropriately when the previously specified conditions occur. This monitoring must be done efficiently since the performance of the DBMS may otherwise become extremely degraded.

(12)

# A DATABASE INTERFACE TO WAND FOR THE NETWORK ALERTER SERVICE

James S. Ribeiro

78-11-02


Department of Decision Sciences
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104



Research supported in part by the Office of Naval Research under Contract N00014-75-C-0462.

408 757

UNIVERSITY OF PENNSYLVANIA

THE MOORE SCHOOL

# A DATABASE INTERFACE TO WAND FOR THE NETWORK ALERTER SERVICE

JAMES STEVEN RIBEIRO

Presented to the Faculty of the College of Engineering and Applied Science (Department of Computer and Information Sciences) in partial fulfillment of the requirements for the degree of Master of Science in Engineering.

Philadelphia, Pennsylvania

December, 1978

Thesis Supervisor

Graduate Group Chairman

ABSTRACT

A DATABASE INTERFACE TO WAND FOR THE
NETWORK ALERTER SERVICE

JAMES STEVEN RIBEIRO

DR. HOWARD LEE MORGAN

The Database Interface (DBI) is responsible for monitoring conditions on a database managed by a particular type of Database Management System (DBMS). At the Wharton School there are several large databases which are managed by WAND, a CODASYL-like DBMS. This thesis outlines the design and implementation of a DBI for WAND.

The DBI will be integrated into the Network Alerter Service (NAS), which is being designed and implemented as a general user service for ARPANET users. The NAS will allow the monitoring of databases at various sites on the ARPANET for conditions of interest to the user.

The WAND DBI is capable of efficiently monitoring these conditions (or alerters) and responding appropriately when the previously specified conditions occur. This monitoring must be done efficiently since the performance of the DBMS may otherwise become extremely degraded.

ACKNOWLEDGEMENTS

The author wishes to thank Stan Cohen and Rishiyur Nikhil for their helpful comments concerning this work. Special thanks are due to Dr. Howard Lee Morgan for his advice and for originally suggesting the topic for this work.

TABLE OF CONTENTS

# LIST OF FIGURES

## 1.0 Introduction

Within highly volatile shared databases many conditions can arise which could be of interest to the database users. These conditions are a result of the large number of transactions made to the database. Alerting is the ability to monitor databases for these conditions of interest and to perform some action whenever such conditions become true. In an alerting system, this is accomplished by allowing the user to define alerters. An alerter consists of a name, a condition to be evaluated, and some corresponding action to be taken whenever the condition is met. An alerting system provides the user with the capability of defining and evaluating these alerters.

For example, in a database containing information about a ship's status (SHIPSTATUS record), an alerter called LOWFUEL could be created to monitor fuel reserves below 60% capacity. The alerter would then notify the user if this condition occurs and specify the ship's hull number and the present fuel reserves. The alerter name is LOWFUEL , the alerting condition is FUELRESERVE LT .60, and the corresponding action is the message stating the ship's null number and the current fuel reserves. This example illustrates the fact that alerters change the concept of a database management system (DBMS) from that of a passive system to an active one. Previously, the user would have to repeatedly enter the query of interest since the DBMS would

only respond when queried.

In this thesis I intend to detail the design and implementation of an alerting Database Interface (DBI) for the Network Alerter Service (NAS) [1]. The DBI is designed for a DBMS oriented to the manipulation of large, network structured files. This type of DBMS was described by Codasyl's Data Base Task Group (DBTG) in their April 1971 report [3]. This particular DBI was written for the Wharton Alerting Network Database (WAND) [7]. WAND is an implementation of a CODASYL-like database management system in use on Wharton's DECSYSTEM-10 and is available commercially in a version called SEED [12]. The WAND DBI operates in conjunction with the WAND system and provides it with rich alerting facilities.

## 2.0 Prior Research

In 1970, Morgan [9] published a paper introducing the concept of alerters in the context of Management Information Systems. In that early paper, alerters were described as interrupts which are used to notify the supervisor of the occurrence of certain conditions. These interrupt generating conditions are actually Boolean conditions on data-items in the database and cause the supervisor to run certain programs.

Much of the work in this thesis is based on the ideas of Buneman and Morgan [2]. They classify alerters into three levels based on the complexities of their conditions. These three major levels are simple, structural, and complex alerters.

1. Simple alerters monitor conditions which can be defined in terms of a single record class. Simple alerters can monitor for any of the following:

    1. The addition, deletion, or modification of a record occurrence to a set of records.

    2. The addition, deletion, or modification of a single field for some record type. An example would be: "Notify me if an employee's age exceeds 65," where age is a field within an employee record.

3. The addition, deletion, or modification of a computed item within a record. An example is the request: "Report any ship's hull number if the number of sick personnel plus the number of damaged equipment units exceeds 10," where number of sick personnel, number of damaged equipment units, and the ship's hull number are all within the ship record.

2. Structural alerters monitor conditions which involve more than one record class and require knowledge of the structure of the database. An example of a structural alerter is the request: "Report a ship's hull number if the number of damaged equipment units exceeds 20 or the number of signtings is over 5," where the SIGHTINGS record is a member of the set owned by the SHIPS record.

3. Complex alerting involves a more global view of the database. The two major classes of complex alerters are statistical alerters and alerters which involve time.

   1. Statistical alerters involve monitoring statistical quantities such as average or median. An example would be the request: "List any department when the average employee's age exceeds 50". Monitoring for

this alerter would require knowledge of the values for all employee's age within the PERSONNEL record.

2. An example of an alerter involving time is the request: "Tell me when a ship has not had a sighting during the last four hours".

Buneman and Morgan also suggest guidelines for implementing simple alerters without continuously polling the database.

Guidlines for adding alerters to the database would be:

1. An incoming alerter should first be checked for consistency with the database schema i.e. valid field names, and field names must be related to the proper record types. Error messages are sent if any inconsistencies arise.

2. An index is established for each (record class, field) pair within the condition of the alerter. At this point the alerter has been successfully defined and stored within the database.

For evaluating alerters:

1. If a modifying update is to be performed, a copy of the old record is made in a temporary area.

2. Once the update is completed a copy of the updated record is placed in a temporary area.

3. If a modifying update was performed, compile a list of the changed field names.

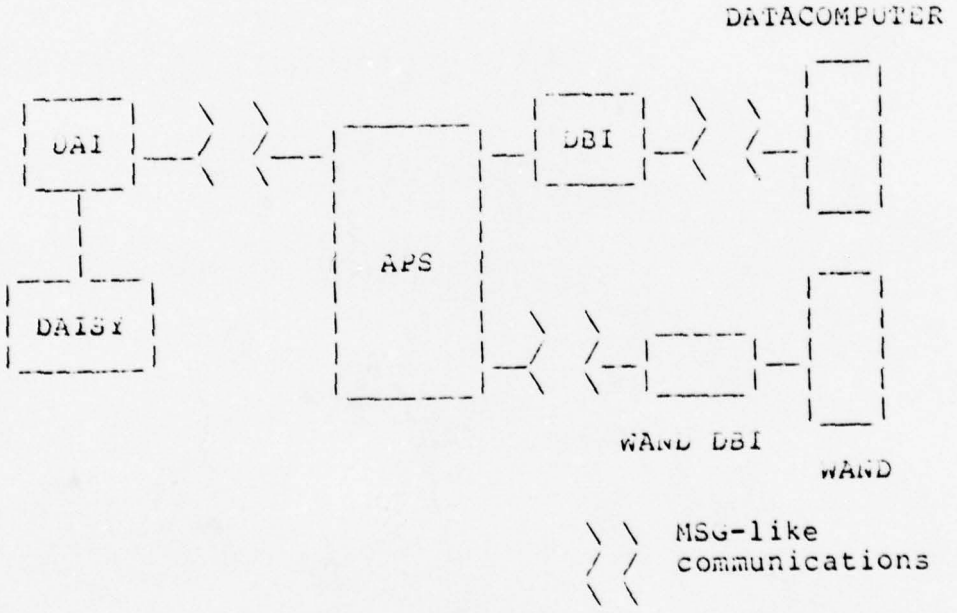
4. Evaluate any alerter indexed under the appropriate record class and field name. The alerter is evaluated by examining the previously stored definition of the condition with respect to the values of the records in temporary storage.

Several alerting database systems have already been constructed at Wharton. Cohen [4] implemented in LISP a system LDEMON which monitors changes in a database through event-driven procedures called demons. Cortes [5] implemented a system for WAND which could monitor changes to fields within the database. Although his system was extremely limited, Cortes used a WAND database to store and retrieve information about alerters. Thus he made extensive use of the CODASYL structures and DML routines available in the WAND system. Kimball [6] implemented the DATA system which performed alerting on a database containing time ordered lists of transactions.

## 3.0 The Network Alerter Service

The Network Alerter Service (NAS) is designed as a general user service for ARPANET users. The overall structure of the NAS is shown in Figure 1.

Figure 1: NAS Structure



The NAS accepts requests from users to monitor one or many databases for the occurrence of a specified event. The User Alerter Interface (UAI) translates from the specific man/machine interface (e.g., DAISY [16]) to a set of messages to be sent to the Alert Processing System (APS). The APS maintains the database of alerters and translates user queries into a set of conditions, each member of which refers only to fields on a particular database.

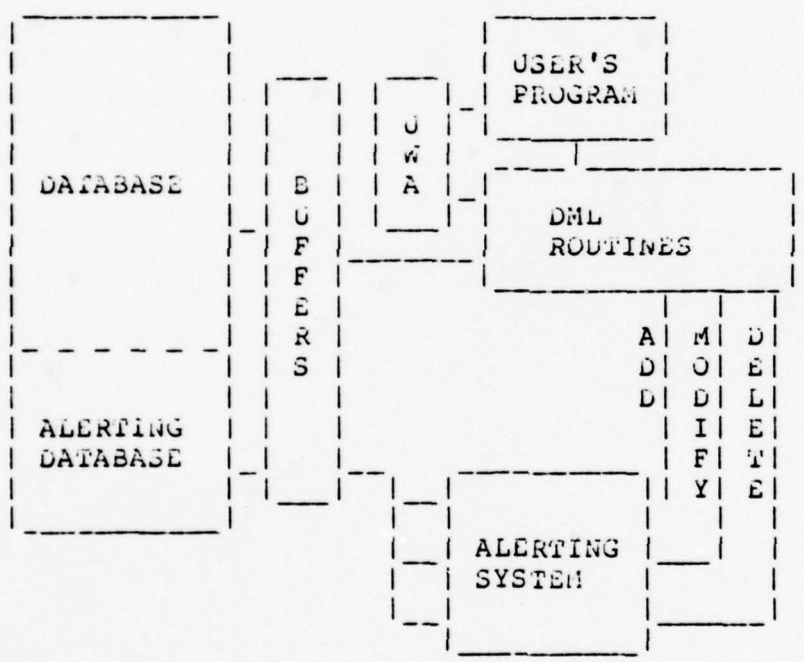
The DBI performs the alerting functions for a particular database. The APS breaks up the user's alerter into pieces, each of which involves only single fields within a single database, and then sends these pieces to the appropriate DBI. The DBI allows the NAS to easily accommodate a new DBMS since adding a new DBMS only involves programming a new DBI for that database.

Seven message types are used for communications between the APS and the DBI. The seven message types are detailed in Appendix A.

## 4.0 Alerting in WAND

An alerting system was implemented for WAND to allow it to handle the message types to and from the APS. Figure 2 depicts the WAND system and its interaction with the alerting system.

Figure 2: Interaction between WAND and the Alerting System



This system allows the complete definition of alerters to be stored in the WAND database. Presently, WAND will only handle simple alerting, although the system has been designed so as to allow more complex alerting at a future date.

Several changes were made to WAND to allow it to interact with the alerting system.

1. The FDP (File Definition Processor) now accepts the additional clause ALERTED DATABASE in the Schema Entry. This clause indicates that the database being defined will be used for alerting. If the FDP parses this clause then the schema is marked for alerting and a file is opened containing special records and sets and they are added to the schema. The Data Definition Language (DDL) of this alerting structure is listed in Appendix B and the complete WAND schema DDL is contained in Appendix C. Notice that the alerting DDL is contained within a separate area, making it invisible to the WAND user. This special structure is used by the alerting system to store definitions of alerters into the database. If the ALERTED DATABASE clause was not specified when the schema was created then a new schema and sub-schema must be created with this clause included in the Schema Entry. The database does not have to be reloaded.

2. A description of every update is sent to the alerting system. This information is passed anytime a record is being added, modified, or deleted in an alerted database. The following information is passed to the alerting system:

   1. Type of update (Modify, Delete, or Add).

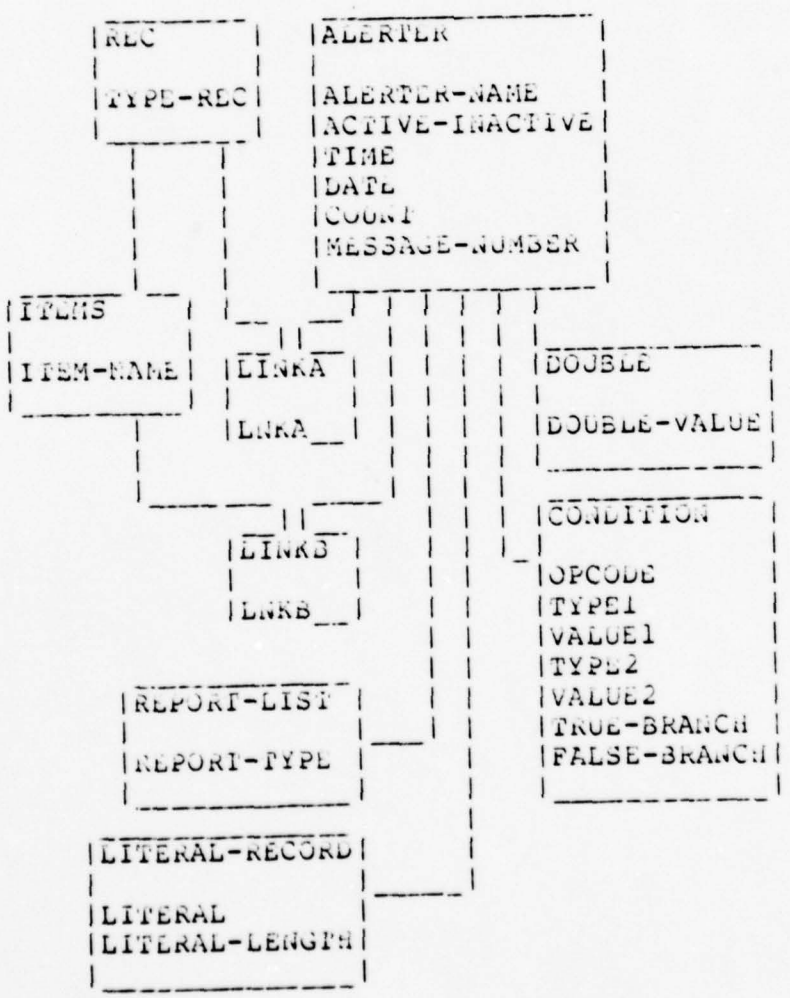
   2. The name of the record being updated.

   3. A copy of the record. If a modifying update is performed, then two record copies must be passed to the alerting system. One copy contains information the record had before being modified and the other contains the current modified record.

## 4.1 Alerting Structure

Figure 3 depicts the structure used by the alerting system to store the definitions of alerters into the database.

Figure 3: Alerting Structure



These records are stored into the database whenever the DBI receives an ADDALERT message from the APS. As discussed in Appendix A, the ADDALERT message has the form:

(ADDALERT <alerter name> <message number> <alerter type> <condition> <report list>)

The various fields of the alerting records contain the following:

1. ALERTER record

    1. ALERTER-NAME is the alerter name passed by the ADDALERT message. The alerter name is checked for uniqueness by first issuing a FINDC in WAND. If the alerter name is unique then the first 30 characters of the alerter name are stored into ALERTER-NAME.

    2. ACTIVE-INACTIVE determines if this alerter is active or inactive. This field is initially set to 1 (active) since alerters are active upon being defined.

    3. TIME is the current clock time. This field is used by the STATUS option. TIME is updated whenever this alerter is triggered.

4. DATE is the current date. DATE is utilized by the STATUS option. This field is updated whenever this alerter is triggered.

5. COUNT is used by the STATUS option to signify the number of times this alerter has been triggered. COUNT is initialized to 0 and is incremented every time this alerter is triggered.

6. MESSAGE-NUMBER is the message number passed by the ADDALERT message and is used in the response back to the APS.

2. REC record

1. TYPE-REC contains one field which is a concatenation of the name of the record which the alerting condition involves, and the alerter type. Alerter type is obtained from the ADDALERT message. The record name is obtained either directly from the alerting condition of the ADDALERT message or implicitly from field names within the alerting condition. This field is used by the alerting system whenever an update is performed. As mentioned in Section 4.0, the alerting system receives the name of the record being updated and the

type of update (ADD, MODIFY, or DELETE). The alerting system then uses these values as a key, and by issuing a FINDC determines if there are any alerters defined for this record and update type. If there is no match then control returns to WAND. Otherwise, all active alerters are evaluated.

3. ITEMS record

   1. ITEM-NAME contains names of fields and is only filled when the alerter type is MODIFY. An instance of this record is stored for each different field name contained in the alerting condition. If a modifying update is performed then a list of affected (changed) fields is compiled. ITEM-NAME is then used to access only those alerters which involve these affected fields.

4. CONDITION record

   1. OPCODE will contain codes for the following operations:

      1. exponentiation
      2. multiplication

3. division
4. addition
5. subtraction
6. equal to
7. not equal to
8. greater than
9. greater than or equal to
10. less than
11. less than or equal to

These are all codes for binary operations. Unary operations are converted to the appropriate binary operations.

2. TYPE1 contains the following codes for the first argument type:

1. Integer field
2. Real field
3. Character field
4. Double precision field
5. Integer field, prefixed by NEW:
6. Real field, prefixed by NEW:
7. Character field, prefixed by NEW:
8. Double precision field, prefixed by NEW:
9. Integer field, prefixed by OLD:
10. Real field, prefixed by OLD:
11. Character field, prefixed by OLD:

12. Double precision field, prefixed by OLD:
13. Integer literal
14. Real literal
15. Character literal
16. Double precision literal
17. Pop value from stack

3. VALUE1 will contain some value for the first argument depending on the type of the argument. If the argument is a field then the value will be the field's offset into the database. If the argument is an integer or real literal then VALUE1 will contain the actual literal value. If the argument is either a character or double precision literal then VALUE1 will contain a pointer into the LITERAL-RECORD or DOUBLE record, respectively.

4. TYPE2 will contain the same information as TYPE1, but for the second argument.

5. VALUE2 will contain the same information as VALUE1, but for the second argument.

6. TRUE-BRANCH will contain the value which represents the position within a set of the next record to access if the evaluation of the current expression is true. This field is used to avoid evaluating redundant expressions.

7. FALSE-BRANCH is similar in function to TRUE-BRANCH except it is utilized if the evaluation of the current expression is false.

5. REPORT-LIST record

1. REPORT-TYPE contains a value for the type of argument within the report list section of the ADDALERT message. The values that can be found in this field are the same as those within the TYPE1 and TYPE2 field.

2. REPORT-VALUE contains a value for the report-list argument depending upon its type. The values that can be found in this field are the same as those within the VALUE1 and VALUE2 field.

6. LITERAL-RECORD record

1. LITERAL contains character strings which can be up to 50 characters in length.

2. LITERAL-LENGTH contains the length of the character string found in LITERAL.

7. DOUBLE record

    1. DOUBLE-VALUE contains an actual double precision value.

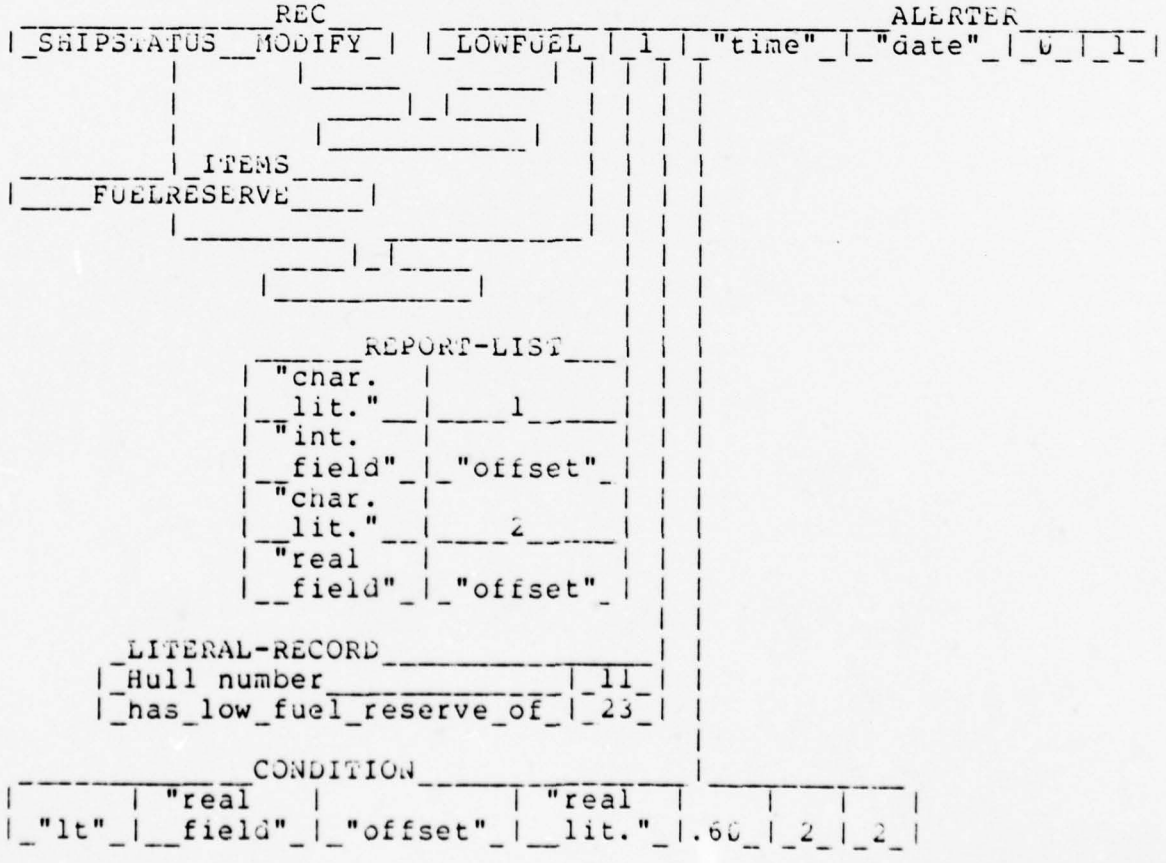
## 4.2 Storing alerters into the database

As an example of the way alerters are stored into the database, reconsider the alerter LOWFUEL from Section 1.0. Assume that alerter is received as:

```
(ADDALERT <LOWFUEL> <1> <MODIFY> <FUELRESERVE LT .60> <'Hull
number',HULLNUMBER,'has low fuel reserve of',FUELRESERVE>)
```

Figure 4 depicts the contents of the alerting records as they would appear in the database. Fields enclosed by quotes (") would actually contain codes for the quoted word.

Figure 4: Storing Alerter LOWFUEL

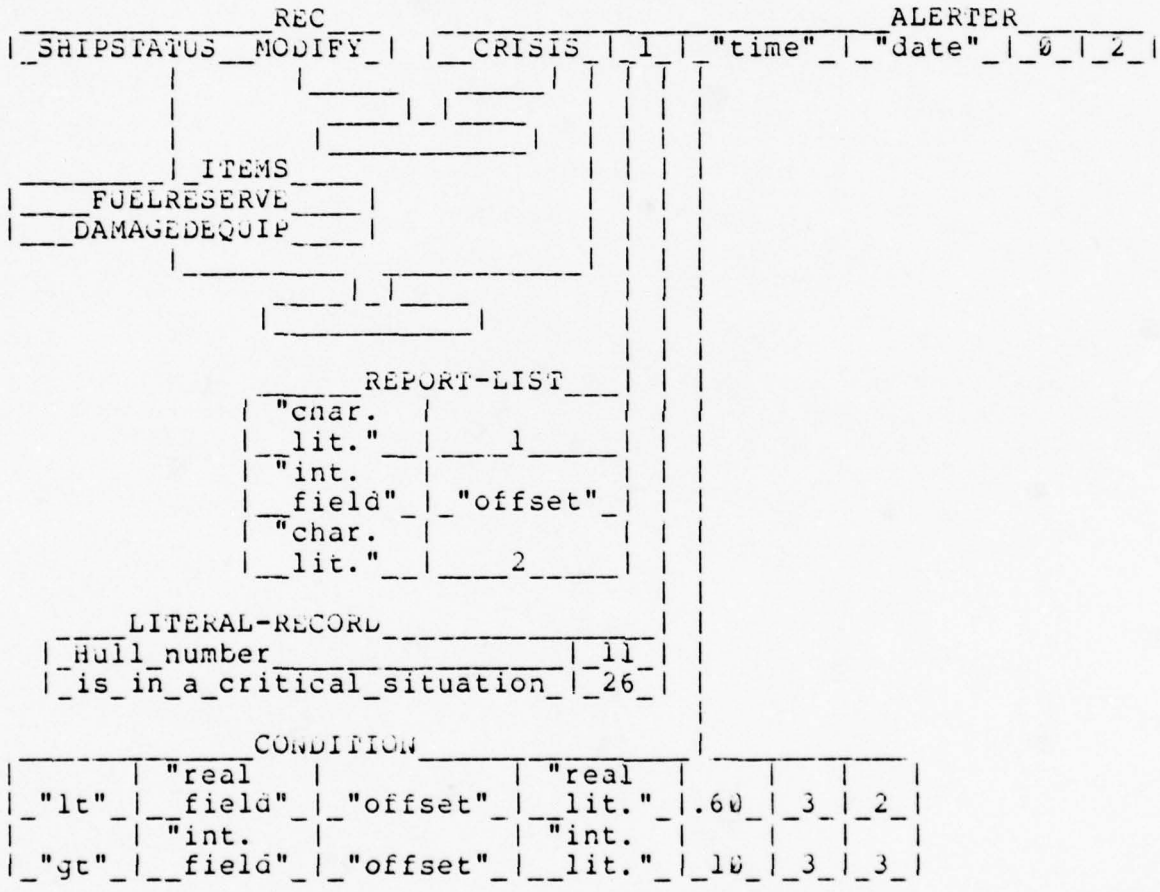


Now assume the following alerter is received:

(ADDALERT <CRISIS> <2> <MODIFY> <FUELRESERVE LT .60 OR DAMAGEDEQUIP GT 10> <'Hull number',HULLNUMBER,'is in a critical situation'>)

Figure 5 depicts the alerter CRISIS as it would be stored in the database.

Figure 5: Storing Alerter CRISIS



## 4.3 Manipulating the database of alerters

As mentioned in Appendix A, the WAND DBI accepts several options via the CHANGESTATUS message from the APS. These options either manipulate or return information about the database of alerters. A functional description of the actions performed by the alerting system to process these options or commands follows.

The ACTIVATE command will set the ACTIVE-INACTIVE field of the appropriate ALERTER record to 1 (active). The ALERTER record is found by issuing a FINDC command using the specified alerter name. ACTIVATEALL finds and activates all ALERTER records. DEACTIVATE and DEACTIVATEALL work similarly but they set the ACTIVE-INACTIVE field to 0 (inactive).

LIST simply finds and lists all the alerter names for all ALERTER records.

The DELETE command first finds the ALERTER record occurrence associated with the specified alerter name. Then the WAND command DELALL is used to delete the current alerter record and all record occurrences that are member occurrences of set occurrences owned by the ALERTER record. Therefore, the current alerter record and the appropriate CONDITION, LITERAL-RECORD, REPORT-LIST, and DOUBLE records will be deleted. Finally, any occurrences of the records

ITEMS and REC are deleted if they are only involved with the deleted alerter. These records cannot be blindly deleted since fields and records may be involved with more than one alerter. DELETEALL simply deletes the entire database of alerters.

The STATUS command first finds the ALERTER record occurrence associated with the specified alerter name. The values from the TIME, DATE, COUNT, and ACTIVE-INACTIVE fields are then returned to the APS.

## 5.0 Conclusions

The motivation for this work was to provide WAND with powerful alerting facilities and to integrate this system into the NAS. At this date the WAND DBI has not been integrated into the NAS. However, a version of the alerting system can be used outside the NAS environment. This version, DBALERT [11] has been successfully demonstrated in the past. A demonstration of DBALERT is listed in Appendix D. Currently, DBALERT is being integrated into a system built by CTEC [6] which simulates updates to the ONRODA database.

As mentioned earlier, the performance of the alerting system must be efficient or the performance of the DBMS may become extremely degraded. No attempt was made to measure the performance of the alerting system . However, DBALERT performs remarkably well and is usually made to "sleep" for several seconds during demonstrations since the alerters are triggered too rapidly. A typical alerting system's efficiency will depend to a great extent on the host DBMS. Approximately 10% of the Fortran statements comprising the alerting system are calls to WAND. Since these calls execute Fortran statements within WAND, I would estimate that 70% or more of the statements executed by the alerting system occur within WAND itself. Therefore, an inefficient DBMS will certainly result in an inefficient alerting

system.

The alerting system implemented can be extended in several ways. One such extension would be the incorporation of time. The alerting system could then be instructed to monitor the database at specified intervals. The system would then be capable of monitoring for a condition such as: "Tell me if the ship Philadelphia has not had a sighting for three hours." Another extension would be the ability to monitor for structural alerters. I believe certain classes of structural alerters could be monitored by extending the WAND DBI. Monitoring for structural alerters is easier in WAND since WAND is aware of its structure and can find unambiguous virtual items within the database. Although the WAND DBI is only capable of monitoring simple alerters it has proven useful since many typical alerters only involve a single record class.

# REFERENCES


1. Buneman, O.P., H.L. Morgan, and S.F. Cohen, Network Alerter Service: Preliminary Design, Working Paper No. 77-07-03, Depatment of Decision Sciences, The Wharton School.

2. Buneman, O.P., and H.L. Morgan, "Alerting Techniques in Database Systems", Proceedings of IEEE Compsac Conference, Chicago, 1977.

3. CODASYL, Data Base Task Group, April 1971 Report. Association for Computing Machinery.

4. Cohen, S.F., Alerters on Network Databases, M.S. Thesis, Moore School, U. of Pa.

5. Cortes, R., An Alerting System for a Data Management System, M.S. Thesis, Moore Scool, U. of Pa.

6. CTEC Inc., "An ONRODA Scenario for Demonstrating Alerts", Falls Church, Va.

7. Gerritsen, R., J. Ribeiro, R. Cortes, and R. Zowader, WAND User's Guide, Working Paper No. 76-01-03, Department of Decision Sciences, The Wharton School.

8. Kimball, K.A., The DATA System, M.S. Thesis, Moore School, U. of Pa.

9. Morgan, H.L., "An Interrupt Based Organization for Management Information Systems," Comm. ACM 13, 12(December 1970).

10. Morgan, H.L., "DAISY: An Applications Perspective," Proceeding of the Wharton/ONR Conference on Decision Support Systems.

11. Ribeiro, J.S., DBALERT: An Alerting System for WAND, Working Paper No. 78-12-02, Department of Decision Sciences, The Wharton School.

12. SEED Reference Manual, International Data Base Systems Inc., Philadelphia, Pa.

# APPENDIX A; MESSAGES

1. The ADDALERT message from the APS to the DBI has the following elements:

    1. A required alerter name. If the alerter name is not specified by the user then it is assigned by the APS. Alerter names in the WAND DBI must be unique and are from 1 to 30 characters in length.

    2. A message sequence number for the message. This is used in the acknowledgement message back to the APS.

    3. An alerter type, which specifies the particular type of monitoring to be performed. Alerter type ADD causes an alerter to be triggered upon adding a record which satisfies the specified condition. DELETE causes an alerter to be triggered when a record with the condition is deleted and MODIFY triggers an alerter whenever any data which is modified satisfies the specified condition.

4. A condition which the DBI is monitoring. A condition can contain literals, field names, arithmetic operators, relational operators, and logical operators.

Field names may be prefixed with optionally specified time qualification (OLD: or NEW:), where OLD: and NEW: refer to the pre and postupdate values of an item in a record whose update triggers an alert.

A literal can be an integer, a real number, a double precision number, or a character string. Character strings must be enclosed within quotes (').

The following arithmetic operators are acceptable.

| | |
|---|---|
| ** | exponentiation |
| * | multiplication |
| / | division |
| + | addition (infix) |
| | or designation of sign (unary) |
| - | subtraction (infix) |
| | or negation (unary) |

These operators are used to combine numeric data-items and numeric literals into arithmetic

expressions. Exponentiation has the highest precedence, followed by multiplication and division (which have equal precedence), then unary plus and negation (which have equal precedence), and finally addition and subtraction (which also have equal precedence).

The relational operators consist of:

| | |
|---|---|
| EQ (=) | equal to |
| NE | not equal to |
| GT (>) | greater than |
| GE | greater than or equal to |
| LT (<) | less than |
| LE | less than or equal to |

Relational operators combine data-items, literals, or arithmetic expressions into relational expressions. Relational operators are of equal precedence and are lower in precedence than all arithmetic operators.

The WAND DBI accepts the logical operators AND, OR, and NOT. Logical operators are used to combine relational expressions and logical expressions into logical expressions. Logical operators are lower in precedence than both arithmetic operators and relational operators.

The NOT operator has precedence over AND and then OR.

A condition is evaluated in the order of precedence of the operators. If there is more than one operator at the same precedence level then they are evaluated from left to right. Parentheses can be used to change the order of evaluation.

A condition can also be the name of a record. In this case, the WAND DBI will monitor the database for additions, deletions, or modifications to the specified record.

5. A report list which consists of the list of values to be reported back when the alerter is triggered. Note that it may contain constants, which the user's own programs may process upon receiving the ALERT message. The report list may also contain the keywords TIME and DATE which report the time and date the alerter was triggered.

2. The ADDED message is sent from the DBI to the APS in response to the ADDALERT message. The ADDED message contains the following items:

1. The alerter name of the alerter.

2. The message sequence number of the ADDALERT message.

3. Flags. Some of the possible values for the flags are: SET, NOTSET, ERROR (item doesn't exist, database down, etc.) or other information, e.g. (checked every n minutes, only updated n times/hour).

3. The CHANGESTATUS message is sent by the APS to the DBI to alter the monitoring status of an alerter. Its elements are:

1. The alerter name.

2. A message sequence number.

3. Options, which will be sent as a list of (<optionname><optionvalue>) pairs or (<optionname>). The WAND DBI accepts the following options:

1. ACTIVATE alerter-name: Activate the specified alerter. An alerter is active on being created. All alerters can be activated by the option ACTIVATEALL.

2. DEACTIVATE alerter-name: Silenced the specified alerter. DEACTIVATEALL will make all alerters inactive.

3. DELETE alerter-name: Delete the specified alerter. DELETEALL deletes all defined alerters.

4. LIST. This returns the names of all the user's alerters.

5. STATUS alerter-name: Return the status of the specified alerter.

The following information is provided by the STATUS command:

1. The number of times this alerter has been triggered.

2. The date this alerter was last triggered.

3. The time this alerter was last triggered.

4. Whether this alerter is inactive or active.

4. The CHANGED message, which is sent by the DBI to the APS, consists of the following:

1. The alerter name.

2. The message sequence number of the corresponding CHANGESTATUS message.

3. Flags, which report error conditions and status changes.

5. The ALERT message is sent by the DBI to the APS when an alerter has been triggered. The elements of the ALERT message are:

1. The name of the alerter which has been triggered.

2. A message sequence number for the ALERT message.

3. The result list which is a list of values corresponding to the report list requested by the ADDALERT message which set up this alerter.

4. Options, which would send additional information, such as the clock time of the update which triggered the alert, the name of the user who entered the update, and other such information, when available.

6. The QUERY message, from the APS to the DBI, requests data to be sent from a particular database. It has the elements:

   1. Message sequence number of the QUERY message.

   2. Record type of the record to be retrieved.

   3. Record identifier or key.

   4. Report list.

7. The ANSWER message is the response from the DBI to the APS in reply to the QUERY message, with the elements:

   1. Message sequence number of the corresponding QUERY message.

   2. Flags which report error conditions.

   3. Result list.

# APPENDIX B; ALERTING DDL

```
AREA NAME IS ALERT-AREA.


RECORD NAME IS LITERAL-RECORD
        LOCATION MODE IS VIA ALERT-LITERAL
        WITHIN ALERT-AREA
LITERAL TYPE IS CHAR 50
LITERAL-LENGTH TYPE IS FIXED.

RECORD NAME IS DOUBLE
        LOCATION MODE IS VIA ALERT-DOUBLE
        WITHIN ALERT-AREA
DOUBLE-VALUE TYPE IS DOUBLE PRECISION.

RECORD NAME IS REC
        LOCATION MODE IS CALC USING TYPE-REC
        DUPLICATES ARE NOT ALLOWED
        WITHIN ALERT-AREA
TYPE-REC TYPE IS CHARACTER 35.

RECORD NAME IS ALERTER
        LOCATION MODE IS CALC USING ALERTER-NAME
        DUPLICATES ARE ALLOWED
        WITHIN ALERT-AREA
ALERTER-NAME TYPE IS CHARACTER 30
ACTIVE-INACTIVE TYPE IS FIXED
TIME TYPE IS CHAR 5
DATE TYPE IS CHAR 10
COUNT TYPE IS FIXED
MESSAGE-NUMBER TYPE IS FIXED.

RECORD NAME IS ITEMS
        LOCATION MODE IS CALC USING ITEM-NAME
        DUPLICATES ARE NOT ALLOWED
        WITHIN ALERT-AREA
ITEM-NAME TYPE IS CHARACTER 30.

RECORD NAME IS LINKA
        LOCATION MODE IS VIA ALERT-REC
        WITHIN ALERT-AREA
LNKA TYPE IS FIXED.
```

```
RECORD NAME IS CONDITION
        LOCATION MODE IS VIA ALERT-COND
        WITHIN ALERT-AREA
OPCODE TYPE IS FIXED
TYPE1 TYPE IS FIXED
VALUE1 TYPE IS REAL
TYPE2 TYPE IS FIXED
VALUE2 TYPE IS REAL
TRUE-BRANCH TYPE IS FIXED
FALSE-BRANCH TYPE IS FIXED.

RECORD NAME IS REPORT-LIST
        LOCATION MODE IS VIA ALERT-REPORT
        WITHIN ALERT-AREA
REPORT-TYPE TYPE IS FIXED
REPORT-VALUE TYPE IS FIXED.

RECORD NAME IS LINKD
        LOCATION MODE IS VIA ALERT-ITEMS
        WITHIN ALERT-AREA
LINKD TYPE IS FIXED.

SET NAME IS ALERT-COND
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS LAST
        OWNER IS ALERTER
        MEMBER IS CONDITION
        LINKED TO OWNER.

SET NAME IS REC-ITEMS
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS LAST
        OWNER IS REC
        MEMBER IS ITEMS
        LINKED TO OWNER.

SET NAME IS ALERT-REC
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS LAST
        OWNER IS ALERTER
        MEMBER IS LINKA
        LINKED TO OWNER.
```

```
SET NAME IS ALERT-REPORT
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS LAST
        OWNER IS ALERTER
        MEMBER IS REPORT-LIST
        LINKED TO OWNER.

SET NAME IS REC-ALERT
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS LAST
        OWNER IS REC
        MEMBER IS LINKA
        LINKED TO OWNER.

SET NAME IS ITEMS-ALERT
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS LAST
        OWNER IS ITEMS
        MEMBER IS LINKB
        LINKED TO OWNER.

SET NAME IS ALERT-ITEMS
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS LAST
        OWNER IS ALERTER
        MEMBER IS LINKB
        LINKED TO OWNER.

SET NAME IS ALERT-LITERAL
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS LAST
        OWNER IS ALERTER
        MEMBER IS LITERAL-RECORD
        LINKED TO OWNER.

SET NAME IS ALERT-DOUBLE
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS LAST
        OWNER IS ALERTER
        MEMBER IS DOUBLE
        LINKED TO OWNER.
```

APPENDIX C; SCHEMA DDL

| SYMBOL | MEANING |
|---|---|
| __ (underline). | WORD MUST APPEAR |
| () | PHRASE MAY BE OMITTED |
| {} | ONLY ONE OF THE LINES MAY BE USED |

Lower case words are replaced by a user-defined name or value.

```
SCHEMA NAME IS schema-name
   (PRIVACY LOCK IS password)
   (DATABASE SIZE IS integer (DYNAMIC) PAGES)
   (PAGE SIZE IS integer WORDS)
   (ALTERED DATABASE)
   (MAXIMUM OF integer RECORDS PER PAGE).


AREA NAME IS area-name
   (AREA SIZE IS integer (DYNAMIC) PAGES)
   (PAGE SIZE IS integer WORDS).


RECORD NAME IS record-name
   LOCATION MODE IS
      {VIA set-name                                                    }
      {CALC USING item-name-1 (DUPLICATES ARE (NOT) ALLOWED)           }
      {DIRECT                                                          }

   (WITHIN area-name).


item-name-2 TYPE IS
          {CHARACTER integer-1}
          {FIXED             }
          {REAL              }
          {DOUBLE PRECISION  }

   (OCCURS integer-2 (BY integer-3)(BY integer-4) TIMES).
```

```
SET NAME IS set-name
   MODE IS {CHAIN (LINKED TO PRIOR)}
           {POINTER-ARRAY          }

   ORDER IS
           {FIRST }
           {LAST  }
           {NEXT  }
           {PRIOR }
           {SORTED}

   OWNER IS {record-name-1}
            {SYSTEM       }

   MEMBER IS record-name-2 (MANDATORY) (AUTOMATIC)
                           (OPTIONAL ) (MANUAL   )
      (LINKED TO OWNER)

       {ASCENDING }
      ({DESCENDING} KEY IS item-name-1

                          {FIRST}
         (DUPLICATES ARE  {LAST } ALLOWED)   )
                          {NOT  }

      (SET OCCURRENCE SELECTION IS THRU
         {CURRENT OF SET          }
         {LOCATION MODE OF OWNER}
            (ALIAS FOR item-name-2 IS data-name)   ).
```

# APPENDIX D; DEMONSTRATION

This Appendix presents an actual terminal session with DBALERT. The database being monitored is the SYSFIL database. The SYSFIL database contains information about jobs running on Wharton's DECSYSTEM-10.

```
.run dbalert

Welcome to B.07 DBLOOK and SEED.  Type HELP or ? for info.

*       watch

Which database would you like to watch? SYSFIL

*       * Look at SYSREC record in SYSFIL database

*       sysrec

RECORD NAME IS SYSREC
    WITHIN AREA1
    LOCATION MODE IS CALC USING JOBNUMBER
      DUPLICATES ARE NOT ALLOWED.
CONTAINS THE FOLLOWING ITEMS:
JOBNUMBER
USERNAME
TTY
PROGNAME
PROJECTNUMBER
PROGRAMMERNUMBER
*       * Now define some alerters

*       addalert

Alerter name

*       monitor_buneman

Alerter type

*       ifmod
```

```
Condition
*       username eq 'BUNEMA' and progname eq 'PASCAL'
Report list
*       'Peter Buneman is using Pascal'
*       addalert monitor_myself ifadd
Condition
*       username eq 'RIBEIJ'
Report list
*       'Jim Ribeiro is now on the system running' progname
*       addalert
Alerter name
*       monitor_nikhil
Alerter type
*       ifadd
Condition
*       username eq 'NIKHIL'
Report list
*       'Nikhil is running' progname
*       * Now lets update the SYSFIL database
*       update
Enter frequency of checking (integer seconds) : 30
Total monitoring time ? (integer minutes): 5
```

```
ALERT MONITOR_MYSELF
Jim Ribeiro is now on the system running
DBALER

ALERT MONITOR_MYSELF
Jim Ribeiro is now on the system running
GHOST

ALERT MONITOR_NIKHIL
Niknil is running
POP10

*       exit

STOP

END OF EXECUTION
CPU TIME: 2.33  ELAPSED TIME: 5:16.82
EXIT
```

DISTRIBUTION LIST

Department of the Navy - Office of Naval Research

Data Base Management Systems Project

Defense Documentation
Cameron Station
Alexandria, VA 22314
12 copies

Office of Naval Research
Branch Office, Chicago
536 South Clark Street
Chicago, IL 60605

Office of Naval Research
New York Area Office
715 Broadway - 5th Floor
New York, N.Y. 10003

Dr. A.L. Slafkosky
Scientific Advisor (RD-1)
Commandant of the Marine Corps
Washington D.C. 20380

Office of Naval Research
Code 458
Arlington, VA 22217

Office of Naval Research
Information Systems Program
Code 437
Arlington, VA 22217
2 copies

Office of Naval Research
Branch Office, Boston
495 Summer Street
Boston, MA 02210

Office of Naval Research
Branch Office, Pasadena
1030 East Green Street
Pasadena, CA 91106

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375
6 copies

Office of Naval Research
Code 455
Arlington, VA 2217

Naval Electronics Lab. Cener
Advanced Software Technology Div.
Code 5200
San Diego, CA 92152

Mr. E. H. Gleissner
Naval Ship Research and
Development Center

Computation and Mathematic Dept.
Bethesda, MD 20084

Mr. Kim Thompson
Technical director
Information Systems Division
OP-911G
Office of Chief Naval Operations
Washington, D.C. 20350

Prof. Omar Wing
Columbia University
in the City of New York
Dept. of Electrical Engineering
and Computer Science
New York, N.Y. 10027

Commander, Naval Sea Systems Command
Department of the Navy
Washington, D.C. 20362
ATTENTION: PMS30611

Captain Richard Martin, USN
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York 09501

Captain Grace M. Hopper
NAICOM/MIS Planning Branch
OP-916D
Office of Chief of Naval Research
Washington, D.C. 20350

Bureau of Library and
Information Science Research
Rutgers - The State University
189 College Avenue
New Brunswick, N.J. 08903
ATTENTION: Dr. Henry Voos

Defense Mapping Agency
Topographic Center
ATTN: Advanced Technology Div.
Code 41300
6500 Brookes Lane
Washington, D.C. 20315